MODEL DEVELOPMENT AID WITH APPLICATIONS TO LINEAR PROGRAMMING

BY

P. PRABHAKAR

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JULY, 1984

MODEL DEVELOPMENT AID WITH APPLICATIONS TO LINEAR PROGRAMMING

A thesis submitted
in Partial Fulfilment of the Requirements
for the degree of
MASTER OF TECHNOLOGY

BY
P. PRABHAKAR

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
JULY, 1984

IMEP-1904-M-PRA-MOD

83745

27/7/84
[illegible]

CERTIFICATE

This is to certify that the work entitled, MODEL DEVELOPMENT AID WITH APPLICATIONS TO LINEAR PROGRAMMING, done by Shri P. Prabhakar has been carried out under my supervision and has not been submitted elsewhere for a degree.

[illegible]
(S. Sadagopan)
Assistant Professor
Industrial and Management Engg.
Indian Institute of Technology
Kanpur 208016

ACKNOWLEDGEMENTS

It is with immense pleasure and great respect that I express my deepest sense of gratitude to Dr. S. Sadagopan for his invaluable guidance and encouragement throughout my thesis work. I am indeed grateful to him for moulding me as a person, as a professional and showing me what I am capable of.

I take this opportunity to express my sincere thanks to all members of IME family, for their constant inspiration.

Thanks to all my friends, who made my stay at IIT enjoyable and memorable.

P. PRABHAKAR

CONTENTS

ABSTRACT

The modeler is the person who formulates a problem. The modeler's forms of the problem have certain characteristics. They are symbolic. They are concise. They are understandable. Finally they are general. They are in a form that is easily read and comprehended by humans.

The algorithm is the computational method, that finds optimal solution to the modeler's form. Algorithm's forms are distinctly different from modeler s forms. They are explicit rather than concise. They are convenient rather than understandable. Choice of an algorithm's form depends on the algorithm and its implementation.

Both of the above forms are essential for solving the model. Yet the fundamental characteristics of these two forms are opposite and incompatible. No single means of expression could serve as both a modeler's form and algorithm's form.

The modeler's form must be transformed into an algorithm's form before it is solved or analyzed. Since the modeler's form is written by a person and algorithm's form is read by a computer, translation necessarily entails both humanwork and machine computation. The best approach is to leave almost all of the work of translation to the computer.

Central to such an approach is a computer readable Algebraic Modeling System, which expresses the model in much the same way the modeler does. This work is a modest step to develop such a system for linear programming problems, which is designed to be simple to learn and easy to use. This system allows the user to specify his LP problems in the natural algebraic format.

The other basic features of this system are:

1. It accepts all types of constraints including range constraints.
2. It introduces slack/surplus variables wherever necessary.
3. Variables can be both lower bounded and upper bounded.
4. Variables can be declared as unrestricted.
5. Problems can be solved by either primal or dual algorithm.
6. Objective function can be either minimized or maximized.
7. Problems can be in algebraic format or matrix format.

In nut shell, the user will not be bothered of details of intermediate processing of the model. In its present form this system can process the language, solve the problem and give the result, but some sophisticated features like sensitivity analysis, model editing, model saving etc., can be readily extended.

This kind of Model Development Aids should lead to the more reliable application of the model at lower over all cost.

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION:

In the early days of mathematical modeling, algorithmic capabilities and raw computational power were the main constraints. The size and complexity of models were thus severely limited by both technical and budgetary constraints. But rapid progress in computer technology has alleviated technical and budgetary constraints on the size of the mathematical models for which we can compute numerical solutions.

With phenomenal improvement in computing power mathematicians developed more and more sophisticated models to depict a near real world situation. Massive research and development efforts in the design and implementation of algorithms have yielded some spectacular results. Implementation of algorithms have reached a level of reliability and efficiency which can be hardly surpassed. Spurred by these developments, the scope of mathematical modeling applications has widened and secured a solid foothold for itself in almost every field. Models are used to simulate, predict or plan the behaviour of physical or social systems and, every day thousands of models are pushed through computers.

As models have grown in size our ability to comprehend and control them has diminished rapidly. Most of the model representation could only be operated by the original development team. Further more, despite the strong efforts in program documentation and model representation actual use of these tools other than by the original designers has been minimal [ALE81].

Besides the tedious job of translating the model from modeler's form to algorithm's form, Johannes [JOH78] identifies five other major obstacles before modelers in effectively using the available software tools. They are the following:

1. <u>Model Documentation:</u>

The documentation of large models and their modifications is a major problem. If a project is large and continues for one or two years, the cost of complete documentation grows exponentially. Within a few years, however the value of the available software becomes essentially zero, since people change jobs, and changes to existing versions require extensive setup cost.

2. <u>Model Communication:</u>

A related problem is that of communication of models to interested persons outside the project team. A technical person is needed to bridge the gap. This may prohibit an outsider from even attempting to satisfy their own curiosity with regard to the model.

3. <u>Model Assumptions</u>:

If models cannot be communicated effectively to outside users, there is no way to check the assumptions and data of a model. As no one but the researcher has a complete knowledge of what went into a model, the published outputs may not necessarily reflect the inputs that an outsider assumes.

4. <u>Model Dissemination</u>:

Another negative effect of poor communication of models to outside users is that no effective dissemination of knowledge can take place.

5. <u>Model Interface</u>:

With the primitive technology in modeling software, there is no common interface with the various solution routines, a modeler may want to use for his family of models. As each solution package usually requires different data structures, it becomes both time and money consuming to switch back and forth between solution algorithms.

The heart of the problem is the fact that solution algorithms need a particular data structure and problem representation which may be difficult to comprehend by modelers. At the same time problem representations that are meaningful to modelers are not acceptable to machines. The translation process by modeler is the main source of difficulties and errors. Here lies the bottleneck of model building and model implementation.

The best approach to this bottleneck is to develop a system which accepts the problem in modeler's form and generates the algorithm's form, thus eliminating the need for tedious and error prone human translation.

1.2 LINEAR PROGRAMMING MODELS:

Linear Programming has become a valuable aid to decision making in many fields. Linear programming techniques are widely used to solve a number of military, economic, industrial and social problems. Three primary reasons for its wide use are [PHI76].

1. A large variety of problems in diverse fields can be represented or at least approximated as linear programming models.

2. Efficient techniques for solving linear programming problems are available, and

3. Ease through which data variation (Sensitivity analysis) can be handled through linear programming models.

1.2.1 Standard Form:

In vector notation the standard linear programming problem can be expressed as:

$$\begin{array}{ll} \text{Maximize (Minimize)} & Z = C\,X \\ \text{Subject to} & A\,X = B \\ & X \geq 0; \quad B \geq 0. \end{array}$$

The main features of the standard form are:

1. The objective function is maximization or minimization type.
2. All constraints are expressed as equations.
3. All variables are restricted to be negative.
4. The right hand side constant of each constraint is non-negative.

1.2.2 General Form:

The general form of linear programming problem is:

$$\text{Maximize (Minimize)} \quad \sum_{j=1}^{n} c_j \, x_j$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij} \, x_j \qquad 1 \leq 1 \leq m$$

$$\ell_j \leq x_j \leq u_j$$

ℓ_j and u_j are lower and upper bounds of the variable x_j. Any problem in general form can be converted into standard form and then solution algorithm can be applied to arrive at the solution.

1.2.3 Thesis Outline:

The objective here is to develop a system that can translate the modeler's general form of linear programming problem into algorithm's form and then solve the problem.

CHAPTER II

LITERATURE SURVEY

2.1 ABSTRACT MODEL:

2.1.1 Introduction:

The standard LP formulation has the form

$$\sum_{j=1}^{n} a_{ij}\, y_j \leq b_i, \qquad 1 \leq i \leq m \text{ (constraints)}$$

$$\text{. Maximize} \quad \sum_{j=1}^{n} c_j\, y_j \qquad \text{(the objective function)}$$

But a potential user is not expected to think in terms of a_{ij}. We may conceive of a model verbally by some statements. E.g., A farm manager may conceive his problem in the following way:

1. The total monthly water consumption for all crops must not exceed the monthly water allotment.

2. Maximize total profit from all crops.

Then the above problem can be written as:

$$\sum_{j \,\varepsilon\, \text{CROP}} a_{ij}\, y_j \leq b_i, \qquad i \;\varepsilon\; \text{MONTH}$$

$$\text{Maximize} \quad \sum_{j \varepsilon \text{CROP}} c_j\, y_j$$

Observe that CROP and MONTH are natural set of objects. Here we call them terms which would be defined separately. The abstract mathematical notation is quite concise. However, a statement of the problem in English is quite natural and is easily understood by human beings. The objective is to think for a language which can capture the conciseness of mathematical notation.

The need for abstract model representation stems from the following properties of many real life programming problems.

1. Linear programming variables that fulfil logically similar roles fall into natural groupings; e.g., on a farm natural groupings might be the crops or the tractors.

2. Some of the constraints have a similar structure. Again consider a farm and assume that the water demand and supply are given on a monthly basis. For each month, a constraint is required to express the bounds on water supply in that month, yielding a set of constraints with similar structure.

3. Some aspects of a model change more often than others. In many cases one conceptual model is applied to different sets of numerical values. In other cases some of variables or constraints are slightly changed, whereas the model as a whole remains its structure.

2.1.2 The Terminology:

The abstract model uses the terminology that is defined separately. It includes two syntactically distinguishable types of identifiers as coefficients, those that represent unknown quantities, to which the linear programming solution ultimately assigns values (the usual Linear Programming variables), and those which represent known quantities (constants). Then there is a need to create and maintain a database for associating data values to identifiers that represent unknown quantities in the model.

Thus the terminology and the database can be viewed as the ENVIRONMENT of the abstract model. When the list of names represented by the terminology and the values are provided from the database, the abstract model can be interpreted to yield a specific concrete model that is in a form ready for solution by a Linear Programming system.

2.1.3 The Advantages:

Though the need for complex language structure will arise, the advantages the many fold. For example, if a particular numerical value must be updated (e.g., the amount of land available for farming changes), this can be done without affecting the terminology of the abstract model. The new components can be recombined into a new concrete linear programming model. Similarly if a new crop is added to the list of crops in the terminology, this need not influence the abstract model, although the numerical information usually will have to be added in the database. Another advantage of this kind of system is that the database can be updated regularly and used to generate reports independent of linear programming context.

2.1.4 Implementation:

The goals that influence the high level system and language also affect the implementation decisions that are not distinctly visible to the user. Without bothering much for the implementation environments the process of implementation can be summarized as follows (Fig. 1): [KAT80].

1. The abstract model in machine readable form is compiled alone without any knowledge of the environment (i.e. terminology or the database). The result for each line is a section of code in a programming language. This code requires terminology and data values as input, and in conjunction with some standard system functions will produce a concrete submodel. In addition to the code itself, the data references mentioned in the every line of the model are produced, as well as two sequences of terms those which are used in constants and those which are used in variables.

2. Next, the list of elements associated with the relevant variable sets are obtained from the terminology system.

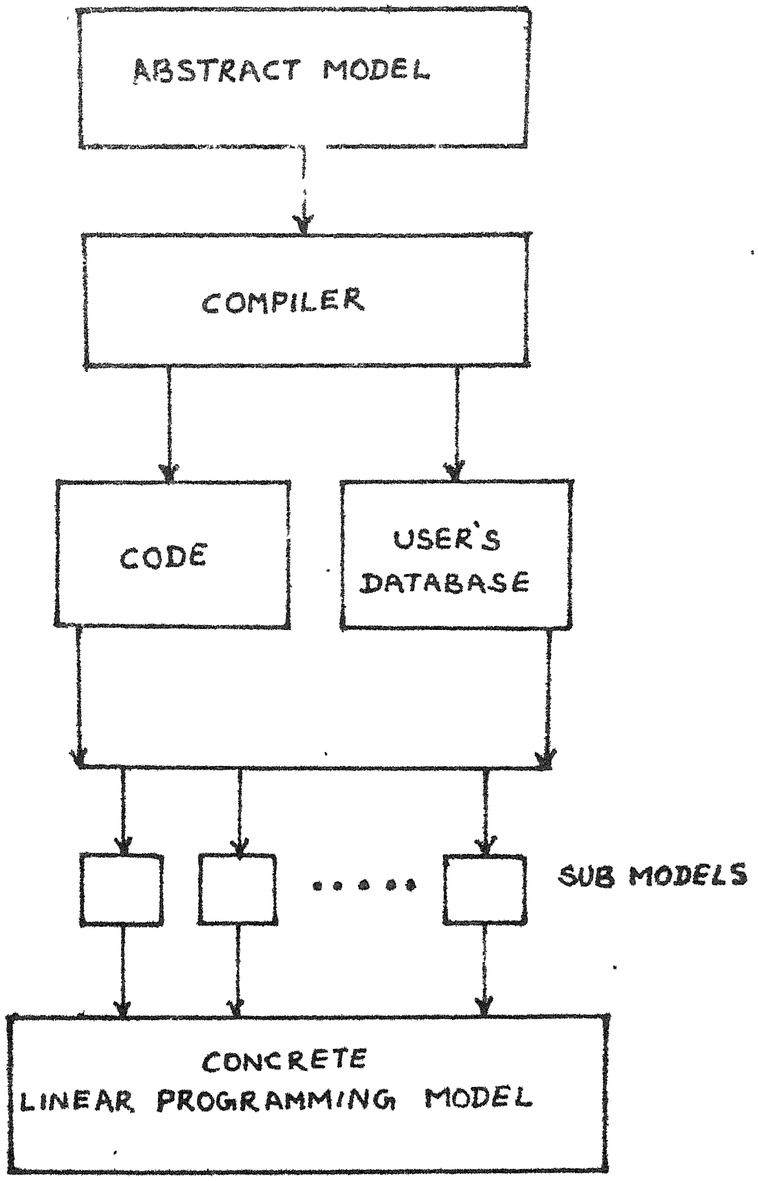

FIG 1. TRANSLATING THE ABSTRACT MODEL

3. The list of elements associated with the terms used in database are founded. These help determine exactly which database values are required.

4. The database is used to obtain the values for the request from 1 using the information from 3.

5. The code generated in 1 is then executed with 2 and 4 as input, producing a collection of independent concrete submodels. Finally the submodels are combined into one large concrete model, where all appearances of the same primitive variables are associated. Later a system for solving concrete linear programming models can be applied.

Note if a single line of the abstract model is changed steps 1 through 5 above for all other lines are unaffected and need not be repeated. If changes are made in the database, steps 1 through 3 are unaffected, and a new terminology leaves 1 unaffected.

With this kind of implementation at heart some systems were developed recently and they were named MATRIX GENERATORS.

2.1.5 The State of Art:

Existing matrix generators include DATAFORM, DATAMAT, GAMMA, IBM MGRW, PERUSE, ANALYZE etc. [ELL82]. Though these systems have quite similar philosophies of design, they differ in details. Each of these systems incorporate its own language in which its MG programs are to be written (Translated version of model into MG FORM). Like higher level programming languages, these MG languages also differ from one another quite distinctively. To convey the model to Matrix Generator the user writes a Matrix Generator programme which can represent the model to MG. Matrix Generator programme has to be written in Matrix Generator language which is typical of FORTRAN, APL etc. Like IBM's ECL (Extended Control Language [SLA78]) can support APL executable statements and PERUSE [KUR80] can support FORTRAN executable statements. In a way these are specially structured languages exploiting the nature of the environment (Appendix III).

2.2 MODELING LANGUAGES:

There is an attractive alternative to matrix generators [FOU83]. It is founded on a simple idea: that modelers should deal with the computer directly in modeler's form. To meet this an LP modeling language is defined by the following two requirements.

1. It must be possible to express any linear program in modeler's form by the use of modeling language.

2. It must be possible to create a computer system that takes modeling language LP as input and that produces a corresponding algorithm's form as output.

2.2.1 General Features:

A modeling language by definition, must satisfy two opposing sets of requirements. One set is imposed by the needs of people, and the other by the nature of the computers.

People want modeling language that is as easy to use and to understand as the familiar modeler's form. Thus a modeling language needs to organize the terminology that modeler's find convenient. The expressions of a modeling language need to be as powerful and flexible as subscripting, indexed sums and other common notation.

Computer systems on the other hand, require a modeling language that can be processed and translated. Both syntax and semantics must be unambiguous and not overly complicated. Unfortunately modeler's forms are ambiguous and complex (as are natural languages generally) and their meanings depend on subtleties of context. Moreover important elements of their notation notably subscripts and signs - are incompatible with ordinary computer hardware.

Nevertheless, an existing modeler's form is a sensible starting point for the design of an attractive and workable modeling language. Such a form need only be modified so that it can be read and processed by Computer. Most importantly the variety and complexity of the existing modeler's form must be reduced so that every expression has an unambiguous syntax and meaning. Notation must be altered to employ a standard character set.

2.2.2 A Prototype:

A hypothetical modeling language XML was designed by Robert Fourer [FOU83]. The correspondence between modeler's form and XML's form have five major sections: sets, parameters, variables, objectives and constraints. Both have entries within sections for individual items (sets, objective, some parameters) and for indexed collection of items (other parameters, variables and constraints).

XML's indexing expressions - such as i over raw ($i \varepsilon$ raw) and t FROM 1 TO time - describe the sets over which collections of items are indexed. The members of indexing sets may be numbers or orbitrary strings. XML incorporates set expressions that specify unions, intersections complements of sets and logical expressions that may include or exclude certain set members (for instance, i OVER Prod, j OVER prod when $1 \neq j$). By use of these set and logical expressions, it is possible to specify particular subsets of the data over which variables or constraints are to be indexed.

XML s arithmetic expression have a powerful indexed sum ('SIGMA') notation that corresponds to the algebraic use of Σ.

There are various concession to the standard character set, such as SIGMA for and x[j]for x_j. As with the modeler s form, the XML model describes all of the set and parameter data symbolically. At a minimum, the XML description lists all of the sets and parameters that the model requires and specifies how certain parameters must be indexed over the sets; certain sets or parameters may also by explicitly defined in terms of others to clarify the formulation or simplify expression.

2.2.3 Implementation of Modeling Language:

ML like languages for linear programming have mostly been developed in early eighties. The more sophisticated of these languages incorporate some kind of symbolic indexing, so that they can conveniently represent fairly large models. Systems that employ these languages are the following:

a) ALPS: Advanced Linear Programming System

b) GAMS: General Algebraic Modeling System

c) LMC: Linear Modeling Capability

d) LPMODEL: Linear Programming Modeling Language

e) MGG: Matrix Generator Generator

f) UIMP: User Interface for Mathematical Programming.

Notably, most of the desirable aspects of an ideal modeling language are represented among these seven systems. All of their languages resemble modeler's form to some degree and do not involve programming. All allow a row-wise (constraint wise) organization and all use subscripted identifiers. They offer various features like: algebraic notation, general linear expressions, symbolic sets, parameters declared independent of data automatic detection of bounds of some kind etc.

These implementations suggest both that modeling language features can be practical to implement and to use and there has been some demand for them. Significantly three of these systems (ALPS, MGG, UIMP) are expressly intended for use with large and complex LP s.

2.2.4 Advantages:

The voids created by matrix generators have been filled by the ML systems. The advantages that accrue are verifiability, modifiability, documentability, independence and simplicity.

a) Verifiability:

First it is necessary to determine whether the modeler's conception of the L.P. is a proper representation of reality. This task the validation of model is independent of computer system.

Second it is necessary to determine whether the ML expression of the LP is a proper representation of the modeler's conception. It is much easier to debug an ML representation than an MG program, since an ML representation is a straight forward and understandable version of an LP. By contrast an MG program represents a fairly complex transformation of an LP.

b) Modifiability:

For an ML user the job of LP modification reduces to determining a new modeler's form. ML modifications closely resemble actual modifications to the modeler's conception of the LP, where as MG modification must be made to a computer program that clearly defers from modeler's conception.

c) Documentability:

Modeler's form documentation can simply be included as part of the modeling language since the ML representation is itself a modeler s form.

d) Independence:

A modeling language describes LP purely in a modeler's form, and gives no indication how they are to be translated, the means of translation and the identity of algorithm's form.

e) Simplicity:

A modeling language saves its user, any concern with intermediate form such as the MG form. An ML user need only be concerned with the modeler's form of LP modeling languages there by simplify the job of linear programming and in doing so they should promote efficiency and reliability of modeling as well.

CHAPTER III

SYSTEM DESIGN

3.1 INTRODUCTION:

Linear Programming has become a valuable aid to decision making in many fields. However, the effort that is required to collect and organize data, to express a Linear Programming (LP) model as a matrix, and to input the matrix of coefficients to the computer, impedes the use of this valuable tool. We have developed a system to help in the process of generating the matrix, given the natural algebraic modeler's form. While good progress has been made in meeting the needs of a less sophisticated user, the process of constructing a model is still complicated.

The increased availability of interactive terminals and the general trend towards providing better programming tools for the user have made it natural to consider systems that concentrate on the construction and development of Linear Programming models, than on their solution. Thus the system developed here, makes Linear Programming more accessible to the user who is not an expert in Computers or Operations Research. This interactive system simplifies the collection and organization of data. It provides a language which frees the user from the necessity of expressing the model as a matrix. The system generates into for a standard LP system, which then actually performs the optimization of the model.

The goal of this work is neither to provide yet another method for solving Linear Programming problems, nor is to provide heuristic guide lines for constructing a Linear Programming model. Rather a declarative language is described, which was designed to be sufficiently expressive, yet simple to learn and use (See Appendix I). The attraction is its naturalness, expressing the model interms natural to the domain of the problem. For the very same reason the system can be a powerful teaching aid also. Note that directly constructing a matrix using any programming languages violates, its naturalness.

3.2 SYSTEM DESIGN:

Besides the general considerations of simplicity and naturalness, some specific properties of Linear Programming models were exploited in the system and language design. A Linear Programming model consists of a set of linear constraints and a linear objective function to be minimized or maximized. The most basic property of such a model is that it is declarative. Another basic property is that various constraints in a model are independent. Thus changing one constraint will have no effect on others, although it does effect the solution to the model.

The system can be bisected into Language processor and Algorithm implementor. The Language processor parses the input, prepares the matrix and sets the options. The Algorithm implementor selects the optional algorithm (Primal or Dual), solves the problem and prints the results. This system can be run either in batch processing mode or in interactive mode. The system features are as follows:

3.2.1 Language Processor:

The most basic property of a linear programming model is that, it is declarative. To declare the model in machine readable form, a set of reserved words are chosen from the modeler's terminology. The reserved words and their actions are discussed in Appendix I.

The system asks for the declaration of variables the user wants to use in his model representation. This is to check the variables in the model representation for the possible spelling mistakes. Variables will be first sorted using Heap sort algorithm [HOR76] and Binary search algorithm [KNU73] is used to check the variables. But in matrix format representation variables will not be sorted and sequential searching will be done to check the variables. The user can also generate variables using the reserved word 'TO'.

Since the LP is a continuous optimization problem, the relational operator '>' is treated as '>=' and the relational operator '<' is treated as '<='.

3.2.2 Input Processing:

The processor can accept the general form of LP (Chapter I) either in algebraic format or in matrix format depending upon the option. The algebraic format of the problem is as follows:

1. Title
2. Var
3. Max (Min)
4. Constraints
5. Bounds
6. Unrestricted
7. End

Though the sequence of the above declarations is not mandatory, variable declaration should come first, before any variable is used in model representation.

The general matrix format of the LP program is as follows:

1. Title
2. Var
3. Max (Min)
4. Matrix
5. Noconstr
6. Constraints type
7. Objective function coefficients
8. Constraint, RHS coefficients
9. End

Due to implementation constraints the sequence from 4 to 8 is rigid.

The processor parses the input, prepares various tables, sets ~~x~~the options and calls for the algorithm implementor.

3.2.3 Algorithm Implementor:

The algorithm implementor has three algorithms. They are:

1. Primal revised simplex [NAR83]
2. Dual revised simplex [NAR83]
3. Primal revised simplex using upper bounding substitution [BRA77]

The first two algorithms cannot tackle upper bounds and are used only when none of the variable is upper bounded. The third one is implemented to tackle upper bounds and is called, only when atleast one of the variables is upper bounded. This is to avoid unnecessary checkings when they are not needed. The algorithm implementor is flexible and any algorithm developed can be coupled without major changes. Once the algorithm solves the problem a print procedure is initiated to output the results.

3.2.4 Output:

Output can be on the terminal or into the file 'OUTPUT'. If the system is run in interactive mode then the output is always on to the terminal. If the system is run in batch processing mode then the results will always be written in the file OUTPUT . Option is available to get the results on to the terminal simultaneously by using the verb 'TALK' in the problem description. When used the system comes up with a prompt '> I am talking, are you listening' and buzzes the bell. System will wait for user response by pressing 'RETURN'. Upon user pressing the 'RETURN' the output will be displayed on to the terminal. The same procedure will be followed in the interactive mode also.

3.2.5 Mode:

The system works in the following two modes depending on the user option:

1. Interactive
2. Batch Processing

In batch processing mode the input is assumed to be in the title 'INPUT'. If input is nonexistent of does not contain the proper input the system gives the message '> NO INPUT file or NO DATA in INPUT file' and stops.

In interactive mode the user enters the problem on the terminal directly and whenever he presses RETURN the system comes up with the prompt > . If any errors are found the system gives the proper message and the user can correct that by keying in afresh. At the end of the problem description he has to type 'END'. Then the system activates the algorithm implementor and solves the problem. After the first problem, the user can key-in the second one, the third one and so on. 'STOP' always brings the user out of the system. Online help facility can be availed by asking 'HELP'.

3.2.6 Online Help:

Online help facility is provided by a brief description of the system and a set of examples. These examples cover all the possible problem representations.

3.2.7 Intermediate Processing:

The system automatically inserts slack and surplus variables wherever necessary. They can be identified with the name -CONSTRn and +CONSTRn respectively where n is the constraint number in the input sequence. The lower bounded variables will be replaced by a corresponding variables. The unrestricted variables will be replaced by a corresponding variables. The unrestricted variables will be replaced by two dummy variables, but while printing the results they will be converted back to the original form. Left hand side restriction in the range constraints is taken as a upper-bounded restriction on the corresponding slack or surplus variable. All this intermediate processing is done in the back ground and the user need not do anything explicitly.

3.3 ILLUSTRATION:

<u>Manufacturer Problem</u>:

Consider the problem of a manufacturer who wishes to determine how many Sofas, Chairs, Desks and Bookcases he should make in order to optimize the use of his resources. These products utilize two different types of lumber labeled Type I and Type II lumber. He has on hand 1500 board feet of Type I and 1000 board feet of Type II. He has 800 man hours available for the total job that must be utilized. His sales schedule requires him to make at least 15 desks and 10 bookcases. Each sofas, chair, desk and bookcase requires 5, 1, 9, 12 board feet of Type I lumber and 3, 2, 5, 10 board feet of Type II lumber. A table requires 2 man-hours to make, a chair 3, a desk 4 and a bookcase 5. The manufacturer makes a total profit of Rs. 12 on a table, Rs. 5 on a chair, Rs. 15 on a desk and Rs. 10 on a bookcase. It is desired to maximize the profit.

3.3.1 <u>The Modeler's Form</u> (For more examples see Appendix II):

The modeler's form of the above problem is stated below:

Maximize: 12 Sofas + 5 Chairs + 10 Bookcases + 15 Desks

Subject to: 5 Sofas : 1 Chair + 9 Desks + 12 Bookcases $\leq$ 15
3 Sofas + 2 Chairs + 5 Desks + 10 Bookcases $\leq$ 10
2 Sofas + 3 Chairs + 4 Desks + 5 Bookcases = 8
Desks $\geq$ 15, Bookcases $\geq$ 10
Sofas, Chairs $\geq$ 0

3.3.2 The Algebraic Input:

The input to the system for the above problem is:

```
VARIABLES

SOFAS, CHAIRS, DESKS, BOOKCASES

MAXIMIZE

12SOFAS + 5CHAIRS + 15DESKS + 10BOOKCASES

CONSTRAINTS

5 SOFAS + CHAIRS   +  9 DESKS + 12 BOOKCASES  < 1500

3 SOFAS + 2 CHAIRS + 5 DESKS + 10 BOOKCASES  < 1000

2 SOFAS + 3 CHAIRS + 4 DESKS + 5 BOOKCASES    =  800

BOUNDS

DESKS  >= 15

BOOKCASES  >  10

END
```

Note the natural algebraic free format used in expressing the problem.

Optimal Value: 3.3729999959E+03

Optimal Solution

Basic variables:

```
CHAIRS      =  8.4000003337E+01
SOFAS       =  2.1899999678E+02
+CONSTR1    =  6.6000018119E+01
```

```
Non-basic variables:

   BOOKCASES     =  1.0000000000E+01

   DESKS         =  1.5000000000E+01

   +CONSTR2      =  0.0000000000
```

3.3.3 The Matrix Input:

The system also accepts the matrix input, since it is convenient for the problems generated by other programs. The matrix input for the above problems is as follows:

```
VAR
SOFAS CHAIRS DESKS BOOKCASES
MAX
MATRIX
NOCONSTR=3
< <  =      !  Constraint type
12  5  15  10! Objective function
 5  1   9  12  1500
 3  2   5  10  1000
 2  3   4   5   800
BOUND
DESKS  >= 15   BOOKCASES  >= 10
END
```

The system output is as follows:

```
OPTIMAL VALUE :   3.3729999959E+03

OPTIMAL SOLUTION
  Basic variables:
     SOFAS        =  2.1899999678E+02
     CHAIRS       =  8.4000003337E+01
     +CONSTR1     =  6.6000018119E+01
   Non-basic variables:
      DESKS       =  1.5000000000E+01
      BOOKCASES   =  1.0000000000E+01
      +CONSTR2    =  0.0000000000
```

observe that the variables are not sorted in this solution contrary to the above solution.

CHAPTER IV

CONCLUSIONS

4.1 SUMMARY:

Mathematical modeling is a potentially powerful tool in the Social Sciences and in Strategic Planning. It will become an effective tool only if the costs involved are reduced significantly, where costs may be measured in monetary units, time and human skills. The key to this all may be found in the communication of models. The most efficient and unambiguous way to communicate models between humans, should also be the way to communicate models with machine. The modeling languages developed are a definite step in this direction.

Although a thorough treatment of the Modeling languages is avoided due to space constraint and complexity of technical details this work demonstrates some of the main features of interest to model builders. The compactness and completeness of a Modeling language is an ever ending refining process, through the feedback of Modeling language users. The struggle to invent systems that can "listen" and "talk" to humans directly, will go on.

4.2 ACHIEVEMENTS:

The system implemented here, is ideal for medium size dense problems. This system is a veritable teaching aid, as the novice can feel both the input and the output. The system helps the beginners in learning it by online help facility and immediate error checking. It relieves the user, even from the burden of converting general form of LP into standard form of LP.

4.3 LIMITATIONS:

The system developed here, accepts algebraic input of a model in standard algebraic format. E.g. The user expresses the constraint $2x + 3y < 100$ directly. Constraints in algebraic form must be of a very limited sort, loosely tied to the matrix formulation. There is no provision for generic constraints and each individual constraint must be input with its explicit numerical coefficients.

The system cannot handle sparse problems efficiently as all the constraint coefficients are stored, rather than storing only nonzero coefficients.

4.4 FUTURE EXTENSIONS:

This work is a modest attempt in implementing a simple and easy to learn system, without losing the naturalness of the LP problem. Besides solving a problem the system can have the following facilities, to increase the flexibility of handling the model and to help the user in learning about the model.

(i) Sensitivity Analysis:

Sensitivity analysis throws lot of insight into the model for variations in the input data. If the system is equipped with sensitivity analysis the user can learn about the model and its reality.

(ii) Model Editing:

Model editing facilities greately enhances the flexibility of some structural changes in the model representation.

(iii) Saving the Model:

After running the model, the user can be given the option of saving its numerical information and the result, in a database for future use. When the user wants to do sensitivity analysis on the model at a later stage, the system need not solve the problem again and can directly start with sensitivity analysis.

REFERENCES

[ALE81] Alexander, M., An algebraic approach to modeling, PROCEEDINGS ON ECONOMIC DYNAMICS AND CONTROL. June 22-24, 1981, Denmark.

[BRA77] Stephen P. Bradely, Arnold C. Hax, Thomas, L. Magnanti, APPLIED MATHEMATICAL PROGRAMMING, Addison-Wesley Pub. Co., 1977.

[FLL82] E.F.D. Ellison and Gautam Mitra, UIMP User interface for mathematical programming. ACM TR. OF MATH. SOFTWARE, Vol. 8, No. 2, 1982.

[FOU83] Robert Fourer, Modeling languages versus matrix generators for linear programming. ACM TR. OF MATH. SOFTWARE, Vol. 9, No. 2, 1983.

[GRE83] Harvey, J. Greenberg, A functional description of ANALYZE. ACM TR. OF MATH. SOFTWARE, Vol. 9, No. 1, 1983.

[HOR76] E. Horowitz and Saratraj, S., FUNDAMENTALS OF DATA STRUCTURES. Computer Science Press, 1976.

[JOH78] Johannes B. and Alexander, M., Towards a general algebraic modeling system. DEVELOPMENT RESEARCH CENTRE, WORLDBANK, 1978.

[KAT80] S. Katz, L.J. Risman and M. Rodeh, A System for constructing linear programming models. IBM SYSTEMS JOURNAL, Vol. 19, No. 4, 1980.

[KNU73] D.E. Knuth, THE ART OF COMPUTER PROGRAMMING, Vol.1, Addison-Wesley, 1983.

[KUR80] W.G. Kurator and R.P. O Neil, PERUSE an interactive system for mathematical programs. ACM TR. OF MATH. SOFTWARE, Vol. 6, No. 12, 1980.

[NAR83] Narsingh Deo, DISCRETE OPTIMIZATION ALGORITHMS WITH PASCAL PROGRAMS, Prentice Hall, 1983.

[PHI76] Don T. Phillips, A.Ravindran, James, J. Solberd, OPERATIONS RESEARCH PRINCIPLES AND PRACTICE. John Willey and Sons Inc., 1976.

[SLA78] L. Slate and K. Speilberd, The extended control language of MPSX/370 and possible applications. IBM SYSTEMS JOURNAL, Vol. 17, No. 1, 1978.

APPENDIX I

THE LANGUAGE

Problems are solved by specifying appropriate control phrases, objective function, constraints and bounds. The system is both interactive and batch processor. As many problems as desired may be input during a given run. Each problem is solved independently of others. If any fatal error occurs during the language processing that problem will be skipped. But subsequent problems will be processed. Note that relational operator '<' is treated as '<=' and '>' is treated as '>='. Now we will describe the language:

1.

Blank cards are ignored and may be inserted for readability.

2. * !

Comments begin with an asterisk (*) or an exclamatory mark (!). These may appear any where in the problem. From that point till the end of that line is considered to be a comment.

3. TITLE

TITLE of the problem

The reserved word TITLE may be used to specify the problem title. The title card should be followed by a card containing the problem title.

TITLE recognizes the first 80 characters only

E.g.

```
TITLE
This problem is referred by DR NARSINGH DEO
```

4. PRIMAL/DUAL

The above card specifies the algorithms to be used. By default it solves the problem xusing revised primal simplex algorithm.

5. VARIABLES or VAR

The reserved word VARIABLES (VAR) specifies the variable list to follow. Individual variables are named on the card by separating each variable name by one or more blanks or by a comma. Ex: LAMPS, BULBS FIRE A sequence of variables may be defined by declaring var1 TO var2 where var1 and var2 have identical prefixes and the suffixes of var1 and var2 are natural numbers. The prefix is defined as leading alpha part and the suffix is defined as the trailing numeric list. X1 TO X5 would define the variables X1, X2, X3, X4 and X5.

6. MAXIMIZE or MAX, MINIMIZE or MIN

The command MAXIMIZE(MAX) or MINIMIZE(MIN) specifies that the objective function is to be either maximized or minimized. MAXIMIZE X1+3X6+X3, MIN 5Y1+6Y3+8.5Y4 The presence of a missing coefficient implies a coefficient of +1. The absence of a variable implies coefficient of zero.

7. CONSTRAINT or CONSTR or CONSTRAINTS:

The phrase CONSTRAINTS(CONSTRAINT,CONSTR) indicates that the constraint specifications are to follow.

```
CONSTRAINTS
4X1 - 5X2  = 8
3X1 - 5X2 >= -10
2X1 - X2 < 20
10 <= 4Y1 - 5Y2 < 40 (Range Constraints)
```

The presence of a variable with a missing coefficient implies a coefficient of +1. The absence of a variable implies a coefficient of zero. Slack and surplus variables will be introduced by the system. +CONSTRn represents surplus variable and -CONSTRn represents slack variable where 'n' indicates the constraint sequence number in model representation.

8. UNRESTRICT or UNRESTRICTED:

This command indicates the variable list whose values are unrestricted.

```
UNRESTRICT
X1,BUDGET
```

Variable declaration with TO is not allowed

```
UNRESTRICT
X1 TO X4    INVALID
```

9. BOUND or BOUNDS:

The phrase BOUND(BOUNDS) is used to fix the lower or upper bounds on any variable name. Default lower bound is zero and default upperbound is infinity. Variable declaration with 'TO' is not allowed.

BOUNDS	BOUND
5 >= X1	
X1 <= 5 VALID	X1 TO X5 >= 5 INVALID
5 <= X1 <= 10	
X1 > 10	

10. INITIAL:

INITIAL calls for a procedure to print or type the initial tableau after converting the problem into standard L.P.

11. EPSILON = 1.0E-5

EPSILON will be followed by a number (between 10E-4 and 10E-8) to be used as the smallest value in problem solving.

12. TALK

TALK enables the results to be typed on the terminal during batch processing.

13. HELP

HELP enables online help facility. To be used in interactive mode.

14. MATRIX

MATRIX calls for a procedure which reads the matrix input. This should precede NOCONSTR.

15. NOCONSTR = n

'n' indicates the number of constraints. Constraint type of all constraints should immediately follow. '>' indicates that the constraint is greater than equal to R.H.S. '<' indicates that the constraint is equal to R.H.S.

```
MATRIX
NOCONSTR = 2
< >
4 2 3     ! Objective function
7 3 6  150!   <=
4 4 5  200!   >=
```

16. END

END indicates the end of the current description. If END is missing between two problems the system skips the former problem, reads the later one and messes up the data giving unpredictable answers.

17. STOP

STOP forces the system to stop after solving the last problem.

See Appendix II for models represented in this language.

APPENDIX II

MODEL REPRESENTATION

Since all these representations are self explanatory. We have not given their modeler's form explicitly.

1.
```
TITLE
D.R. NOTES -- PAGE NO 63
VARIABLES
X1, X2
MAXIMIZE
8X1 + 8X2
CONSTRAINTS
3X1-5X2 >= 30
2X1-X2<= 20
X1 +2X2 <15 !   Treated as <=
END
```

2.
```
dual   !    System does not differentiate LC UC
var    *    Rev dual simplex has been chosen
x1 to x6
constr
4x1 -5x2 +7x3+ x4 =8
-2x1 +4x2-2x3 +x5= -2
x1- 3x2 +2x3 +x6 = 2
```

```
    min
    xl +3x2 +2x3
    title
    This problem was referred by Dr. Narsingh Deo.
    end
3.  VAR
    X1 TO X3
    TITLE
    A problem for unrestricted variables
    MIN
    X1 +X2 +X3
    CONSTRAINTS
    X1 -3x2 + 4x3 = 5
    x1- 2x2 <= 3
    2x2 - x3 >= 4
    UNRESTRICT
     X3
    INITIAL   *  Prints initial tableau
    end
4.  VAR
    X1 TO X3
    MAX
    4X1+2X2 +6X3
    CONSTR
    4X1 -X2 <= 9
    -X1 +X2 + 2X3 <= 8
    -3X1 +X2+4 X3 <=12
    BOUNDS ! Upper and lower bounds specified
    1 <= X1 <=3
    X2<= 5
    X3<= 2
    END
5.  var
    y1 y2
    max
    8y1 + 5y2
    constr
    10 <= y1 +4y2 <=40 ! Range constraints
    15 <=6y1 -4y2 <=60
    bound
    x1 <= 10
    x2 <= 10
    TALK          * Types the results on the terminal
    end
    stop          * Stop after solving this problem
6.  VARIABLES
    . A, B,C
    MAXIMIZE
    4A + 2B + 3C
    CONSTRAINT
    7A + 3B + 6C <= 150
    4A + 4B + 5C < 200
    INITIAL
    EPSILON=1.0E-08
    END

7.  VARIABLE
     A B C
    MAX
    MATRIX   ! Matrix form of the above problem
    NOCONSTR=2
    < <
     4    2     3    ! Objective function.
     7    3     6    150
     4    4     5    200
    EPSILON>1.0E-08
    INITIAL
    END
```

APPENDIX III

AN ABSTRACT MODEL

This appendix illustrates the different forms of LP by use of the following elementary production and inventory problem.

THE PROBLEM:

A factory can manufacture some number of different products over the next, say T production periods. Each product returns a characteristic estimated profit per unit, which varies from period to period. The factory size imposes a fixed upper limit on the total units manufactured per period additionally each product requires fixed characteristic amounts of certain raw materials per unit.

Limited quantities of raw materials must be stored now for use in the next T periods. Each raw material has a fixed characteristic storage cost per unit period. Any material still unused after period T has a certain remaining value. What products should be manufactured in what periods to maximize total expected profit minus total storage costs, adjusted for the remaining value of any unused raw materials

The Modeler s form: (Abstract Model)

Given P set of products

R set of raw materials

and $T > 0$ Number of production periods

$M > 0$ Maximum total unit production per period

$a_{ij} \geq$ For $i \in R$, $j \in P$. Units of raw material i needed to manufacture one unit of Product j.

$b_i \geq 0$ For $i \in R$: Max initial stock of raw material i

c_{jt} For $j \in P$, $t = 1 \ldots T$: estimated profit per unit of product j in period t.

d_i For $i \in R$: Storage cost per period per unit of raw material i.

f_i For $i \in R$: estimated remaining value per unit of raw material i after last period

Define: $x_{jt} \geq 0$ for $j \in P$, $t = 1 \ldots T$: Units of product j manufactured in period t. $S_{it} \geq$ For $i \in R$, $t = 1 \ldots T+1$: Stock of raw material i at the beginning of period t.

Maximize

$$\sum_{t=1}^{T} \left(\sum_{j \in P} c_{jt} x_{jt} - \sum_{i \in R} d_i s_{it} \right) + \sum_{i \in R} f_i s_{i,T+1}$$

Total overall periods of estimated profit less storage cost, plus value of remaining raw materials after last period.

Subject to:

$$\sum_{j \in P} x_{jt} \leq M$$

For $t = 1 \ldots T$: Total unit production per period must not exceed maximum.

$s_{i1} \leq b_i$ For $i \in R$: stock for period one must not exceed maximum.

$s_{i,t+1} = s_{it} - \sum_{j \in P} a_{ij}\, x_{jt}$

For $i \in R$, $t = 1 \ldots T$: Stock for next period equals stock for present period less raw materials used in present period.

Matrix Generator s Form: (Using MaGen Matrix Generator Language):

```
*-------------------------------------------
    DICTIONARY
*-------------------------------------------
    CLASS T     Set of production periods
    1,2,3
    CLASS U     Set of periods preceding production periods
    0,1,2
    CLASS V     Extra period after last production period
        4
    CLASS W     Period preceding extra period
        3
    CLASS PRD Set of products
    LOW         Low quality product
    MED         Medium quality product
    HIH         High quality product
    CLASS RAW Set of raw materials
    SCR         SCRAP raw material
    NEW         new raw material
```

```
*-----------------------------------------------
    DATA
*-----------------------------------------------

    TABLE   M       Maximum total unit production per period
            MAX
    MAX     40
    TABLE   A       Unit of raw material needed to manufacture unit o
            LOW   MED   HIH            each product.
    SCR      5     3     1
    NEW      1     2     3
    TABLE   B       Maximum initial stock of raw material
            MAX
    SCR     400
    NEW     275
    TABLE   C       Estimated profit per unit product in each period
              1     2     3
    LOW      25    20    10
    MED      50    50    50
    HIH      75    80   100
    TABLE   D       Storage cost per unit for each raw material.
            STOR
    SCR     0.5
    NEW     2.0
    TABLE   F       Remaining value for each raw material after last
          VALUE     period
    SCR     15
    NEW     25
```

```
*--------------------------------
    ROWS SECTION
*--------------------------------

    COPY

 NAME   SAMPLE

    FORM ROW ID

    obj = obj

*   Balance production and stock of each raw material in each

*   period

    BAL (RAW)(T) = FIX

*   Limited total production in each period

    CAP(T) = MAX

*--------------------------------
    COLUMNS SECTION
*--------------------------------

    COPY

    COLUMNS

*   Manufacturing activity for each product in each period

    FORM VECTOR X(PRD)(T)

*   Consumption of each raw material by each product in each perio

    BAL(RAW)(T) = TABLE A((ORD),(RAW))

*   Consumption of capacity in each period

    CAP(T) = 1

*   Estimated profit from each product in each period

    OBJ = TABLE C((T), (PRD))

*   Stock piling activity for each raw material

    FORM VECTOR S(RAW)(T)
```

```
*   Production of each raw material from stocks, start of each
*   period
    BAL (RAW)(T) = -1
*   Consumption of each raw material into stocks after each
*   period
    BAL(RAW)(U/T) = 1 EXCEPT T = 1
*   Storage costs for each raw material in each period
    OBJ = -TABLE D(STOR, (RAW))
*   Stock piling activity for each raw material after last period
    FORM VECTOR S(RAW)(V)
*   Consumption of each raw material into stocks after last
*   period
    BAL (RAW)(W/V) = 1
*   Remaining value, for each raw material after last period
    OBJ = TABLE F(VALUE, (RAW))
*---------------------------------
    RHS SECTION
*---------------------------------
    COPY
  RHS
    FORM VECTOR RHSIDE
*   Production capacity in each period
    CAP(T) = TABLE M(MAX,MAX)
```

```
*--------------------------------
    BOUNDS SECTION
*--------------------------------
    COPY
 BOUNDS
    FORM BOUNDS MAX STOCK
*   Limit on stock of each raw material initially
    S(RAW)1, UP = TABLE (MAX,  (RAW))
    END
```

Modeling Language Form: (XML)

```
    SETS
     Prod   COMMENT   Set of products
     raw    COMMENT   Set of raw materials
    PARAMETERS
     time   ATTRIBUTES   Positive, integer
            COMMENT      number of production period
     max    ATTRIBUTES   positive
            COMMENT      maximum total unit production
    bb      INDEXING     OVER raw !  b[I]
     c      INDEXING     OVER prod, FROM 1 TO time !  c[j,t]
     d      INDEXING     OVER raw,  !  d[i]
     f      INDEXING     OVER raw   !  f[i]
 VARIABLES
     X      INDEXING OVER Prod, FROM 1 to time  ! X[j, t]
     S      INDEXING OVER raw,  FROM 1 to time + 1 ! S [i, t]
```

```
OBJECTIVE       ATTRIBUTE       MAXIMIZE
                SPECIFICATION   SIGMA t FROM 1 TO time
                                (SIGMA j OVER Prod (c[j,t]* x[j,t])
                               -(SIGMA i OVER raw (d [i]* s[i,t]))
                               +SIGMA i OVER raw (f[i] * s[i,T+1])
CONSTRAINTS

   limit          INDEXING        t FROM 1 TO time
                  SPECIFICATION   SIGMA j OVER Prod (x[j,t]) <= max
   init           INDEXING        i OVER raw
                  SPECIFICATION  s[i,1] <= b[i]
   bal            INDEXING        i OVER raw, t FROM 1 TO time
                  SPECIFICATION  s[i,t+1] = s[i,t]
                                 -SIGMA j OVER Prod (a[i,j]* x[j,t])
END
```

83715

IMEP-1984-M-PRA-MOD